A Multivalued logic model of planning

M. Baioletti and **A. Milani** and **V. Poggioni** and **S. Suriani** [1]

Abstract. In this work a model for planning with multivalued fluents and graded actions, based on the infinite valued Lukasiewicz logic, is introduced. In multivalued planning, fluents can assume truth values in the interval $[0, 1]$ and actions can be executed at different application degrees also varying in $[0, 1]$. The notions of planning problem and solution plan also reflect a multivalued approach. Multivalued fluents and graded actions allow to model many real situations where some features of the world cannot be modeled with boolean values and where actions can be executed with varying strength which produces graded effects as well. Even if most existing planning models fail to address this kind of domains, our model is comparable with models allowing flexible actions and soft constraints. A correct/complete algorithm which solves bounded multivalued planning problems based on MIP compilation is also described and a prototype implementation is presented.

1 Introduction

Many–valued logics have recently been considered in Software Engineering research because they provide a support for an explicit modelling of uncertainty, disagreement and fuzziness. In many areas of artificial intelligence research, related to AI planning, several extensions of classical models with multivalued approaches have been proposed. In model checking, the classical approach has been extended to a multi–valued framework [4]. In [2] a new framework for soft constraint satisfaction problems (CSP), in which fuzzy CSP and weighted CSP are particular cases, has been proposed. In [10] a many–valued SAT solver based on clauses with multivalued literals has been presented.

In AI planning there exist many real application domains in which the relevant properties cannot be expressed by a crisp boolean value, the actions can be applied with a graded level and the effects can depend on that level. Consider for instance a simple scenario where the action *open_door* is available in the domain and modifies the truth value of the fluent *door_is_open*.

A typical probabilistic planning model would treat both the fluent *door_is_open* and the successful execution of the action *open_door* as uncertain, assigning to them a belief degree in $[0, 1]$. Such degrees are different from the truth value that the real world will assign to the fluent and to the action, which will be definitely 1 or 0, i.e. true/false or successful/unsuccessful. In other words, the planner is uncertain about boolean, but unknown, properties like *door_is_open*. A planner with resources would treat the fluent *door_is_open* as a real function whose initial value has to be measured, for instance by the openness angle. Since the action *open_door* cannot have as a numerical parameter the angle by which *door_is_open* is changed, it is not possible to model the strength of the action. From a multivalued point of view, instead, the truth value of the fluent *door_is_open* denotes a real world property which can vary from 0 (the door is completely closed) to 1 (the door is completely opened). In other words the door can "really" be opened at an intermediate degree, between "completely closed" and "completely opened". Moreover the multivalued action *open_door* can be applied with a "strength" ranging from 0 (not applied at all) to 1 (completely applied). The resulting effect on the fluent *door_is_open* will depend on the application level of action *open_door* and on the previous truth value of the fluent itself, i.e. the effects can also be additive.

Choosing a suitable execution degree of an action a, it is possible to obtain a sufficient level for some fluents, for instance goals or subgoals, while keeping low the side effects of the execution of a.

The use of graded actions is also useful in domains where the execution degree is related to the cost/execution time of the action. Therefore it is possible to define a metric function based on execution degrees by which optimal plans in terms of overall cost/time are generated.

It is worth noticing that many fuzzy concepts cannot be represented by numerical resources, because the fuzzy values could not be obtainable by a measurement process or they could be affected by subjective factors.

In the multivalued/fuzzy logics scenario we preferred to use a logic based on T-norms because of its well–founded mathematical aspects. Among this class of logics we have chosen the Łukasiewicz logic because the semantics of its operators is more suitable for our framework. For these reasons we have excluded other well–known logics as Product Logic, in which for instance the negation operator is not involutive.

In this paper a new framework for a general theory of multivalued planning based on the infinite-valued Łukasiewicz logic [5, 9] is introduced and an algorithm for a multivalued planner is described. The proposed model can be seen as a generalization of the classical planning, since boolean and multivalued actions and fluents can be used in the same domain.

The paper is organized as follows. Some basic concepts on Łukasiewicz logic are recalled in Section 2, the model of multivalued planning is presented in Section 3 and an algorithm solving problems in such a model is proposed in Section 4. Finally an example, related works and conclusions are pointed out in Sections 5 and 6.

2 A brief introduction to Łukasiewicz logic

Łukasiewicz logic is a multivalued extension of classical logic, well known from the theoretical point of view [9, 5]. In this section we will report only some useful definitions. The conjunction operator, $x \odot y = \max(0, x + y - 1)$, has the properties of a generic T–norm: it is commutative, associative, monotonic with respect to all the variables; it has 0 as absorbing element and 1 as unit element.

[1] Università degli Studi di Perugia, Italy, email: {baioletti,milani,poggioni,suriani}@dipmat.unipg.it

The implication is defined as the residual of the conjunction operator and is expressed by $x \rightarrow y = \min(1, 1 - x + y)$.

The negation, which can be defined as $\neg x = x \rightarrow 0$, is expressed as $\neg x = 1 - x$ and is involutive. Using the last property and De Morgan's law it is possible to define a disjunction operator, expressed as $x \oplus y = min(1, x + y)$. This operation is a co–norm: it is commutative, associative, monotonic monotonic with respect to all the variables; it has 1 as absorbing element and 0 as unit element.

Implication and disjunction are related by the condition $x \rightarrow y = \neg x \oplus y$.

It is possible to define an operation similar to subtraction, as $x \ominus y = x \odot \neg y = \max(0, x - y)$. Under some condition on x and y, it is the inverse of additive disjunction . Moreover $(x \ominus y) \ominus z = x \ominus (y \oplus z)$ holds.

Lastly we introduce the lattices operators *min*–conjunction $x \wedge y = x \odot (x \rightarrow y)$ and *max*–disjunction $x \vee y = (x \rightarrow y) \rightarrow y$. They coincide, respectively, with the minimum and the maximum between x and y.

3 The Model

In this section a model of multivalued fluents and graded actions is presented. This model allows to use boolean fluents and classical actions jointly with these new kinds of fluents and actions.

3.1 States

Let $\mathcal{F}$ be the finite set of fluents. A state is described by a function S which assigns to each fluent f a truth value belonging to $[0, 1]$ and denoted by $S(f)$.

It can be seen as a natural extension of the classical state definition in which the range of the state function is $[0, 1]$ instead of $\{0, 1\}$.

The value of a negated fluent is denoted by $S(\neg f)$ and it is computed as $S(\neg f) = \neg S(f)$. In general, given a generic logic formula ϕ, $S(\phi)$ denotes the truth value of ϕ in the state S.

A boolean fluent is a fluent whose possible values are restricted in $\{0, 1\}$.

3.2 Actions

Following the classical action definition introduced in STRIPS, an action a is described by a list of preconditions *pre(a)* and a list of effects *eff(a)*.

Differently from the classical models, an application degree $\alpha(a) \in [0, 1]$ is assigned to each action a. The application degree of a boolean action is only 0 or 1, while for a multivalued action, $\alpha(a)$ can assume any intermediate value.

The degrees 0 and 1 have the same meaning as in the classical model (respectively, the action is not executed and the action is fully executed), while an intermediate value means that the action is executed with a lesser strength.

Informally we can say that each precondition is provided with a threshold. The action is fully executable only when the truth values of each precondition exceed the corresponding thresholds, while in the other cases the action can be executed with a smaller degree.

The execution of an action a can modify the truth value of fluents by means of additive changes: p is a positive effect of a if, after its execution, the value of p is increased, while p is a negative effect if the value is decreased. The amount of increment is proportional to the application degree of the action and can be tuned by a parameter which models the strength of the effect.

3.2.1 Action Preconditions and Action Executability

Definition 1 *Action Preconditions*
The preconditions of an action a are defined by a list of pairs (β_i, p_i), where p_i is a literal (i.e. a fluent or its negation) and β_i is a real number in $[0, 1]$, called the threshold *of p_i.*

The threshold can be read as a *coefficient of importance* of p_i for a or a sort of a hardness degree of the precondition. $\beta_i = 0$ means that the precondition is *totally soft*, i.e. the truth value of the fluent does not affect the executability of a. While $\beta_i = 1$ means that the precondition is *totally hard*, i.e. that the truth value of p_i is required to be 1 for a to be fully executed.

If p_i is a boolean fluent, then β_i must be 0 or 1.

Definition 2 *Action Executability.*
An action a is executable in a state S with application degree $\alpha \in [0, 1]$ if $\alpha \leq \alpha_{max}(a, S)$, where

$$\alpha_{max}(a, S) = \bigwedge_{(\beta, p) \in pre(a)} (\beta \rightarrow S(p)). \quad (1)$$

It is reasonable to compute a maximum application degree because the minimum application degree must be always 0, i.e. the planner can choose whether to apply a and, in this case, it can apply a with any actual application degree smaller or equal to $\alpha_{max}(a, S)$.

The *min*–conjunction $\wedge$ is used in (1), instead of the usual Łukasiewicz conjunction $\odot$, because the latter, differently from the former, is not idempotent and if x and y are smaller than 1, $x \odot y$ is smaller than x and y. This property would cause some problems when there are some preconditions which are strictly related, modelling, for instance, the same feature of the world. In that case it is reasonable to expect that the presence of both the preconditions would not alter the executability of the action with respect to the case in which only one precondition is present.

The expression $\beta_i \rightarrow S(p_i)$ can be seen as a fuzzy extension of the crisp relation $S(p_i) \geq \beta_i$. In fact, if $S(p_i) \geq \beta_i$, then the implication assumes truth value 1, while if $S(p_i) < \beta_i$ then the implication assumes a value lesser than 1. The higher the difference between $S(p_i)$ and β_i is, the lesser the value of the implication is. The use of $\rightarrow$ therefore allows to execute a (with $\alpha(a) < 1$) even if $S(p_i) \leq \beta_i$. Moreover, as desirable, this expression is increasing with respect to $S(p_i)$ and decreasing with respect to β_i.

Finally, it is important to note that a negative precondition has a particular semantics, i.e. it requires that the truth value of the fluent is small. In fact $\beta \rightarrow S(\neg p) = 1$ if and only if $S(\neg p) \geq \beta$, i.e. $S(p) \leq 1 - \beta$.

3.2.2 Action Effects and Action Execution

Definition 3 *Action Effects.*
The effects of an action a are defined by a list of pair (γ_i, e_i), where e_i is a literal and γ_i is a real number in $[0, +\infty)$, called weight *of a over e_i.*

It is not allowed to have an action having both a positive and a negative effect over the same fluent.

Definition 4 *Action Execution.*
In the state S', resulting from the execution of an action a with application degree α in a state S, for each effect (γ, e) of a

$$S'(e) = S(e) \oplus \Pi_\gamma(\alpha) \quad (2)$$

where $\Pi_\gamma(\alpha) = \min(\gamma \cdot \alpha, 1)$.

Since the aim of our model is to deal with actions which modify the world state according to their application degrees, additive effects are more suitable. For instance, if an action is executed with a low application degree, the change on the world has to be small. An action which assigns truth values to fluents depending only on α does not verify this important property.

The unary operator Π_γ allows to reduce or to amplify the effect of the action execution with respect to the application degree and, in particular, reflects the idea of effects proportionally related to the application degree.

It is easy to see that the application of(2) to a negated fluent $\neg f$ yields to $S'(\neg f) = S(\neg f) \oplus \Pi_\gamma(\alpha)$ and, since $S'(\neg f) = 1 - S'(f)$, to

$$S'(f) = S(f) \ominus \Pi_\gamma(\alpha).$$

Therefore, as expected, the action execution decrements the truth value of fluents in its negative effects, while increments the fluents in its positive effects.

If e_i is a boolean literal, then the corresponding γ_i is represented by ∞ because by this value it is possible to simulate assignments to 1 (or to 0).

3.2.3 Multivalued Planning Problems

A *multivalued planning problem* is defined by a quadruple $(I, \mathcal{O}, G, \sigma)$ where I is a truth assignment over the fluents $I : \mathcal{F} \rightarrow [0, 1]$ and denotes the initial state; $\mathcal{O}$ is a set of boolean and multivalued actions; G is defined by a list of pairs (θ_i, g_i) having the same structure of action preconditions, and σ is a real number in $[0, 1]$ called *global threshold*.

A valid plan is an ordered sequence of pairs $\langle a_1 : \alpha_1, a_2 : \alpha_2, \ldots, a_n : \alpha_n \rangle$, where each α_i represents the application degree of action $a_i \in \mathcal{O}$. Moreover, let $S_0 = I$ and S_{i+1} the state resulting from the execution of the action a_i is S_i, each action a_i must be executable in S_i with application degree α_i.

It is clear that non interfering actions could be simultaneously executed. While a boolean concept of interference among actions is natural, it is not straightforward to define a multivalued version of this concept and this point needs further investigations. Therefore in this first model we restrict our attention only to linear plans.

Definition 5 *Solution Plan*
A solution plan is a valid plan where in the final state S_n

$$\bigwedge_{(\theta,g)\in G} (\theta \rightarrow S_n(g)) \geq \sigma \tag{3}$$

Note that a solution plan is a plan which achieves all the goals in a multivalued fashion. First, the θs represent the importance or hardness of each goal, similarly to what the βs represent for the action preconditions. Then, σ specifies a global parameter which represents a minimal satisfaction degree desired for the solution plan.

Since we are solving a planning problem by means of a MIP solver, it is straightforward to allow the definition of a metric function in terms of a linear function of the application degrees of some actions. Then a solution plan must also minimize the metric function.

4 An Algorithm for Multivalued Planning

The idea of the algorithm is to solve bounded multivalued planning problems described in the model presented in Section 3 in three steps. At first, a planning graph [3] is constructed until either the upper bound U for time–steps is exceeded or a state verifying the necessary condition for the solution existence is reached. Then the graph representation is translated into a MIP problem (Mixed Integer Programming), and finally a MIP problem solver is called. If the MIP problem has a solution, then also the planning problem has a solution which is directly derived from the MIP problem solution. If the MIP problem has no solution, the main loop continues by adding a new level until a solution is found or the bound U is exceeded. It is easy to prove that this algorithm is correct and complete in the class of the U–bounded multivalued planning problems.

4.1 Planning Graph Construction

The planning graph built in our system is similar to the planning graph used in Graphplan and other planning systems. The planning graph is a $2T + 1$–leveled graph $G = (V, E)$ whose vertex set is $V = F_0 \cup A_0 \cup F_1 \cup A_1 \cup \ldots \cup F_T$, where F_t is the set of *multivalued fluents* at time $t = 0, \ldots, T$ and A_t is the set of *graded actions* at time $t = 0, \ldots, T - 1$. Each edge in E links either a fluent $f \in F_t$ to an action $a \in A_t$, such that $(\beta, f) \in pre(a)$, or an action $a \in A_t$ to a fluent $f \in F_{t+1}$, such that $(\gamma, f) \in eff(a)$.

Note that there are no edges denoting mutual exclusion relationship, since such a concept is not used in our framework.

In the following, $S_t(f)$ means the truth value of the fluent $f \in F_t$ at the time-step t and $\alpha^*_t(a)$ denotes the upper bound for the application degree α of the action $a \in A_t$.

The values of $S_0(f)$ and $\alpha^*_0(a)$ can be computed in an exact way, because the initial state is completely known and $\alpha^*_0(a)$, for each $a \in A_0$, can be computed directly using the formula 1.

When $t > 0$, since it is not known which action will be executed in each time–step, the truth value $S_t(f)$ for each fluent $f \in F_t$ is unknown and, for the same reason, it is not possible to compute a realistic value of $\alpha^*_t(a)$ for each action $a \in A_t$.

Anyway, it is possible to compute a lower bound $\underline{S}_t(f)$ and an upper bound $\overline{S}_t(f)$ for the value $S_t(f)$ by taking into account the maximum change (in the positive and in the negative cases) caused by any possible action executed at time $t - 1$.

In order to compute a tighter upper bound for $\alpha_t(a)$ which takes into account the fact that in each time–step only one action is executed, we must compute lower and upper bounds for $S_t(f)$ conditioned to the execution at time $t - 1$ of a given action.

Therefore for each fluent $f \in F_t$ and each action a, we define the interval $[\underline{S}^a_t(f), \overline{S}^a_t(f)]$ which provides a bound for $S_t(f)$ after the application of the action a at time $t - 1$. Clearly,

$$[\underline{S}_t(f), \overline{S}_t(f)] = [\min_{a\in A_{t-1}} \underline{S}^a_t(f), \max_{a\in A_{t-1}} \overline{S}^a_t(f)] \tag{4}$$

holds, where

$$\underline{S}^a_t(f) = \begin{cases} \underline{S}_{t-1}(f) & \text{if } (\gamma, \neg f) \notin eff(a) \\ \underline{S}_{t-1}(f) \ominus \Pi_\gamma(\alpha^*_{t-1}(a)) & \text{if } (\gamma, \neg f) \in eff(a) \end{cases}$$

$$\overline{S}^a_t(f) = \begin{cases} \overline{S}_{t-1}(f) & \text{if } (\gamma, f) \notin eff(a) \\ \overline{S}_{t-1}(f) \oplus \Pi_\gamma(\alpha^*_{t-1}(a)) & \text{if } (\gamma, f) \in eff(a) \end{cases}$$

Then, we define $\alpha^*_{bt}(a)$ as the maximum execution degree of the action a, computed considering $[\underline{S}^b_t(p), \overline{S}^b_t(p)]$ as the interval bounding the truth values at time t for each fluent p involved in the preconditions of the action a. In other words, $\alpha^*_{bt}(a)$ is the maximum

execution degree of the action a at the time t taking into account that the action b was executed at time $t-1$.

$$\alpha^*_{bt}(a) = \bigwedge_{(\beta,p)\in pre(a)} (\beta \rightarrow \overline{S}^b_t(p)) \wedge \bigwedge_{(\beta,\neg p)\in pre(a)} (\beta \rightarrow \neg\underline{S}^b_t(p)).$$

Finally, we compute

$$\alpha^*_t(a) = \max_{b\in A_{t-1}} \alpha^*_{bt}(a). \quad (5)$$

It is easy to prove that the computation of $\alpha^*_t(a)$ directly from $\underline{S}_t(f)$ and $\overline{S}_t(f)$, would produce a higher estimate of the maximum application degree.

Note nevertheless that a complete implementation of the previous method is not feasible. In fact the computation of $\alpha^*_t(a)$ at a time t would require the computation of $\underline{S}^{a_0,\ldots,a_{t-1}}_t(f)$ and $\overline{S}^{a_0,\ldots,a_{t-1}}_t(f)$ for every possible sequence of actions $a_0,\ldots,a_{t-1}$.

The first time–step where a solution can exist is the first one which verifies the goal condition

$$\bigwedge_{(\theta,f)\in G} (\theta \rightarrow \overline{S}_t(f)) \ \wedge \bigwedge_{(\theta,\neg f)\in G} (\theta \rightarrow \neg\underline{S}_t(f)) \ \geq \ \sigma. \quad (6)$$

4.2 Constraints Extraction

Since it has been shown, for instance in [8], that constraints expressed by operators of Łukasiewicz logic can be transformed in linear constraints on integer and real variables, the constraints contained in the planning graph built as explained in Section 4.1 can be compiled into a Mixed Integer Programming (MIP) problem.

The truth value of each fluent f and the application degree of each action a at time t are respectively denoted by the real variables x_{ft} and x_{at} which take values in $[0,1]$. For boolean fluent and actions the range is restricted to $\{0,1\}$.

Different kinds of constraint must be satisfied in order to have a solution plan according to the model presented in Section 3. They can be grouped in *action executability rules*, *action execution rules* and *goal satisfiability rules*. Moreover, according to the definition of a valid plan, we have to express a linearity condition for each plan called *plan linearity constraint*.

The number of constraints in the MIP problem will be a polynomial of the number of fluents and actions in the planning graph as shown in the following sections.

4.2.1 Action Executability Rules

At each time–step t, the actual application degree of an action a must be less or equal than the upper bound $\alpha^*_t(a)$. Therefore,

$$x_{at} \leq \min(\min_{(\beta,f)\in pre(a)}\{\beta \rightarrow x_{ft}\}, \min_{(\beta,\neg f)\in pre(a)}\{\beta \rightarrow \neg x_{ft}\}\ ).$$

Therefore for each $a \in A_t$ and for every $t = 0,\ldots,T-1$

$$\begin{cases} x_{at} \leq (1-\beta) + x_{ft} & \text{for all } (\beta,f) \in pre(a) \\ x_{at} \leq (1-\beta) + (1-x_{ft}) & \text{for all } (\beta,\neg f) \in pre(a) \end{cases}$$

The number of constraints added to MIP problem for each time–step is equal to the total number of preconditions of every action.

4.2.2 Action Execution Rules

The value of of a fluent f at a time–step $t+1$, $x_{f,t+1}$, can be computed by the following equation which depends on the value of f at the previous time–step and on the execution of some action.

$$x_{f,t+1} = x_{ft} \oplus \left(\bigoplus_{(\gamma,f)\in eff(a)} \Pi_\gamma(x_{at}) \right) \ominus \left(\bigoplus_{(\gamma,\neg f)\in eff(a)} \Pi_\gamma(x_{at}) \right). \quad (7)$$

Note that since only one action is executed at each time–step, in (7) at most one term among $\Pi_\gamma(x_{a,t-1})$ is greater than zero and affects the value of x_{ft}.

The translation of (7) in MIP constraints is performed by means of the techniques shown in [8] and in [1]. The overall number of constraints added to the MIP problem for each time–step is proportional to the number of fluents.

4.2.3 Goal Satisfiability Rule

This rule derives directly from the goal condition defined in (6) and requires that each subgoal (θ, f) must be fulfilled with a truth value greater or equal to the global threshold. It is expressed by the formula

$$\bigwedge_{(\theta,f)\in G} (\theta \rightarrow x_{fT}) \ \geq \sigma. \quad (8)$$

As shown in subsection 4.2.1, this constraint can be encoded as

$$\begin{cases} (1-\theta) + x_{fT} \geq \sigma & \text{for all } (\theta,f) \in G \\ (1-\theta) + (1-x_{fT}) \geq \sigma & \text{for all } (\theta,\neg f) \in G \end{cases}$$

The number of constraints added to the MIP problem is equal to the number of problem goals.

4.2.4 Linearity Plan Rule

With this constraint we express the condition that only one action can be executed in a given time–step, i.e. at each time t, $\exists! a \in A_t : x_{at} \neq 0$.

This constraint can be represented directly by the MIP system

$$\begin{cases} x_{at} \leq \lambda_{at} & \text{for all } a \in A_t \\ \sum_{a\in A_t} \lambda_{at} = 1 & \\ \lambda_{at} \in \{0,1\} & \text{for all } a \in A_t \end{cases} \quad (9)$$

It is easy to see that this condition is true if and only if at each time–step t there exists at most an action a, such that $x_{at} \neq 0$.

The number of constraints added to the MIP problem for each time–step is equal to the number of actions plus one.

4.3 Implementation

A prototype implementation of the previously described algorithm has been made in C++, using as a MIP solver the open–source library *lpsolve*, which is not very efficient, although the preliminary results obtained on small examples are encouraging.

We expect that further code optimizations and the use of more efficient linear programming libraries, like CPLEX, will produce good results from the computational point of view.

Another different approach to solve multivalued planning problems without metric is a compilation into a Linear Arithmetic Logic problem, handled by some solvers like MathSAT [6].

5 An Example

In order to model domains and problems with multivalued fluents and graded actions some obvious extensions to PDDL3 were added.

In this small example we present a domain having generic objects that can be wet and must to be painted. There are also some grippers that can be wet and hold objects. These domain properties are represented by the following fluents: *(obj_wet o), (painted o), (gr_wet g), (holding o g)* where *o* is an object and *g* is a gripper.

This domain has four actions. *dry* is a graded action that can dry a free gripper. *paint* is a graded action that can paint an object held by a gripper. *pickup* and *putdown* are boolean actions that respectively pick an object up from the table and put an object down to the table. The property to be on the table is represented by *(on_table o)*. A gripper can hold only one object, can pick an object up only if it is free and the object is on the table. It can put an object down only if it holds it. A gripper can be wet and can be safely used only if it is not too much wet.

This example cannot realistically be modeled by classical planning domains, because qualitative properties like "be wet" or "be painted" cannot suitably be represented by boolean propositions: a gripper could be "a little wet" or "just damp". Soft preconditions allow, for instance, to execute the action *pickup(gr,obj)* even if the gripper is not totally dry. Moreover, the classical planning model is not adequate for an action like *dry(gr)* because it would be applied with the same strength, both when the gripper is *very wet* and when it is *a little wet*. These are the action descriptions:

```
(:action paint
   :type graded
   :parameters (?obj - object ?gr - gripper)
   :precondition (and (1 (holding ?obj ?gr))
   :effect (0.8 (painted ?obj)) )
(:action dry
   :type graded
   :parameters (?gr - gripper)
   :precondition (0.8 (gr_wet ?gr))
   :effect (0.9 not(gr_wet ?gr)) )
(:action pickup
   :type boolean
   :parameters (?obj - object ?gr - gripper)
   :precondition (and (1 (free ?gr))
           (1 (on_table ?obj))
           (0.7 (not(gr_wet ?gr))) )
   :effect (and (∞ not(free ?gr)) (∞ (not(on_table ?obj)))
                (∞ (holding ?obj ?gr)) ))
(:action putdown
   :type boolean
   :parameters (?obj - object ?gr - gripper)
   :precondition (1 (holding ?obj ?gr))
   :effect (and (∞ (free ?gr)) (∞ (on_table ?obj))
                (∞ (not(holding ?obj ?gr))) ))
```

An example of a problem is presented. We suppose to have the goal to paint the objects minimizing the `dry` application degrees in all the plan.

```
(define (problem example)
  (:domain slippery_gripper)
  (:objects gr1-gripper gr2-gripper obj1-object obj2-object)
  (:init (= 1 (free gr1)) (= 1 (free gr2))
         (= 1 (ontable obj1)) (= 1 (ontable obj2))
         (= 0.2 (gr_wet gr1)) (= 0.5 (gr_wet gr2))
         (= 0.2 (obj_wet obj1)) (= 0.7 (obj_wet obj2))
         (= 0.3 (painted obj1)) (= 0.6 (painted obj2))  )
  (:goal (and (0.8 (painted obj1))(0.7 (painted obj2))) )
  (:threshold 0.9)
  (:metric minimize (is-executed dry))
)
```

6 Related Works and Conclusions

One of the mainly related planning models is the non–Boolean approach introduced in [11]. In this model *propositions* denoting fuzzy relations and *flexible operators* are allowed. The main differences with our approach is that they use finite–valued fluents and the mapping between preconditions and effects is obtained by a set of disjoint conditional clauses, where each clause determines the satisfaction degree of the effects associated with it. Despite of the fuzziness of the approach, the activation of a conditional clause is crisp, i.e. in flexible planning slight differences in the satisfaction degree of preconditions can activate completely different set of effects. Finally the concept of graded action is not present in [11], where operators are applied only at fixed application degree, because only the satisfaction degree of preconditions determines the effects of the action. In our approach, instead, the planner decides which application degree to assign to an action among the available values.

It is easy to see that some types of resources can be encoded in our model by appropriate multivalued properties values and additive effects. For instance the quantitative resource of *fuel* in a tank, varying between 0 and 100 litres, can be modeled by a multivalued fluent *tank_full* in [0,1] which represents, in some sense, a fuzzy normalization of the resource *fuel*. Operators which produce/consume *fuel* would increment/decrement the truth value of fluent *tank_full*.

Recently PDDL [7] has been extended in order to allow *soft* constraints and goals preferences; the planning problem consists then in finding an admissible solution which maximizes the preference function. It is worth noticing that, although preferences are not considered in our approach, they could be encoded by introducing explicit multivalued *preference fluents* into the operators, and by specifying in the goals the desired preference level.

An interesting direction for future work is to study the effects of building the planning model on a finite–valued logic.

Different and more efficient approaches to solve multivalued planning problems need to be considered, such as graph based algorithms or using mixed boolean–real solvers.

Finally, from an experimental point of view, it is necessary to develop a set of benchmark problems for significant domains involving graded actions in order to test the performance of the implementation.

REFERENCES

[1] E. Balas, 'Disjunctive programming.', *Annals of Discrete Mathematics*, **5**, 3–51, (1979).

[2] S. Bistarelli, U. Montanari, and F. Rossi, 'Semiring-based constraint satisfaction and optimization', *Journal of ACM*, **44**(2), 201–236, (1997).

[3] A. Blum and M. Furst, 'Fast planning through planning graph analysis.', *Artificial Intelligence*, **90**(1–2), 279–298, (1997).

[4] M. Chechik, B. Devereux, A. Gurfinkel, and S. Easterbrook, 'Multi–valued symbolic model-checking', *ACM Transactions on Software Engineering and Methodology.*, **12**(4), 1–38, (2003).

[5] R.L.O. Cignoli, I.M.L. D'Ottaviano, and D. Mundici, *Algebric Foundations of many–valued Reasoning.*, Kluwer Academic Publisher, Dordrecht, Boston, London, 2000.

[6] Marco Bozzano et al., 'The mathsat 3 system', in *Proc. CADE-20*, (2005).

[7] A. Gerevini and D. Long. Plan constraints and preferences in pddl3. Tech. Rep., Dep. of Electronics for Automation, University of Brescia, Italy, August 2005.

[8] R. Hähnle, 'Proof theory on many–valued logic – linear optimization – logic design: Connections and interactions.', *Soft Computing*, **1**(3), 107–119, (1997).

[9] P. Hajek, *Mathematics of Fuzzy Logic.*, Kluwer Academic Publisher, Dordrecht, Boston, London, 1998.

[10] C. Liu, A. Kuehlmann, and M. Moskewicz, 'Cama: a multi–valued satisfiability solver', in *Proc. of IEEE/ACM Int. Conf. on CAD*, pp. 326 – 333, (2003).

[11] I. Miguel, P. Jarvis, and Q. Shen, 'Efficient flexible planning via dynamic flexible constraint satisfaction.', *Engineering Applications of Artificial Intelligence*, **14**(3), 301–327, (2001).